

Monitoring and Discovery in a Web Services Framework: Functionality and Performance of the Globus Toolkit's MDS4

Jennifer M. Schopf^{1,4}, Mike D'Arcy³, Neill Miller², Laura Pearlman³, Ian Foster^{1,2}, Carl Kesselman³

1. Mathematics and Computer Science Div., Argonne National Laboratory
2. Department of Computer Science, The University of Chicago
3. Information Science Institute, University of Southern California
4. UK National eScience Centre

Abstract

The Globus Toolkit's Monitoring and Discovery System, (MDS) defines and implements mechanisms for service and resource discovery and monitoring in distributed environments. We introduce here MDS4, the new monitoring and discovery system component included in Globus Toolkit version 4. MDS4 is distinguished from previous similar systems by its extensive use of interfaces and behaviors defined in the new WS-Resource Framework and WS-Notification specifications, and by its deep integration into essentially every component of the Globus Toolkit.

We describe the MDS4 architecture and the relevant Web Service interfaces and behaviors to allow users to discover resources and services, monitor resource and service states, receive updates on current status, and visualize monitoring results. We also describe how MDS4 can be used to implement large-scale distributed monitoring and distributed systems, and present preliminary experimental results that provide insights into the performance that can be achieved via the use of these mechanisms.

1 Overview

In a Grid environment, the set of resources available for use by a virtual organization can change frequently – new resources and services (compute servers, file servers) may be added; old ones may be removed; capacity may be increased or decreased; and basic properties of a resource or service may change, for example, a data server may be upgraded to one with larger capacity, different access rates, and different access protocols. Because these systems are so dynamic in nature, *discovery* – the process of finding suitable resources to perform a task – can be a significant undertaking. Similarly, *monitoring* – the process of observing resources or services to track their status for purposes such as fixing problems and tracking usage – can be more complicated in Grids because of the dynamic, distributed nature of these environments.

Typical monitoring and discovery use cases include providing data so that resource brokers can locate computing elements appropriate for a job, streaming data to an application steering application so adjustments can be made to a running application, and notifying system administrators when changes in system load or disk space availability occur in order to identify possible performance anomalies.

The Globus Toolkit's solution to these closely related problems is its Monitoring and Discovery System (MDS): a suite of components for monitoring and discovering Grid resources and services on Grids [CFF+01]. MDS4, the version recently released as a part of the Globus Toolkit 4 [Foster05], uses standard WSRF interfaces to provide query and subscription interfaces to

arbitrarily detailed resource data and a trigger interface that can be configured to take action when pre-configured conditions are met. The services included in MDS4 acquire their information through an extensible interface that can be used to query WSRF services for resource property information, execute a program to acquire data, or interface with third-party monitoring systems

Grid computing resources and services can advertise a large amount of data for many different use cases. MDS4 was specifically designed to address the needs of a Grid monitoring system – one that publishes data that is of use by multiple people across multiple administrative domains. As such, it is not an event handling system, as is NetLogger [GT03], or a cluster monitor in its own right, as is Ganglia [MCC04], but can interface to more detailed monitoring systems and archives, and can publish summary data using standard interfaces.

In this paper, we detail MDS4 services, infrastructure, data sources, and its visualization tool. In Section 3 we then give preliminary performance results.

The principal contributions of this paper are as follows:

- We show by example how monitoring and discovery capabilities can be integrated into the design of a distributed computing infrastructure so that any and every resource and service can be monitored and discovered in a uniform manner.
- In so doing, we validate the value of primitive interfaces and behaviors defined by the WSRF and WS-N specifications as a basis for building such systems.
- We present performance results that provide insights into the performance of our MDS4 implementation of a WSRF/WSN-based monitoring and discovery system, and compare those results to previous non-WSRF/WSN based systems.

2 MDS4 Details

MDS4 builds heavily on capabilities provided by the WSRF and WS-Notification specifications [FCF+05]; indeed, it can be viewed above all as a use case for those specifications, which define the mechanisms used to describe information sources, access information via both queries and subscriptions, and manage information lifetimes.

In the following sections we describe the basics behind the MDS4 implementation, beginning with the Web Service standards that underpin our approach. Two higher-level services are described in Section 2.2: an Index Service, which collects and publishes aggregated information about grid resources, and a Trigger Service, which collects resource information and performs actions when certain conditions are triggered. These services are built upon a common infrastructure called the *Aggregation Framework*, described in Section 2.3, that provides common interfaces and mechanisms for working with data sources. MDS4 also includes several software components, called *Information Providers*, described in Section 2.4, that are used to collect information, and a web-based user interface called *WebMDS*, described in Section 0. We describe a typical MDS4 deployment in Section 2.6.

2.1 Web Services Standards Used By MDS4

Grid computing resources and services can advertise a large amount of data for many different use cases. Our previous experience with a mixed-protocol toolkit made it clear that the best way to leverage a monitoring infrastructure was to have basic interfaces and monitoring functionality as part of every service in a standard way. In this way basic monitoring and discovery data would become part of the core of every service, not an exception to the rule. Based on this experience,

several Web Services standards have emerged to address the interfaces for interacting with service data, including registration, querying, and naming:

- WS-ResourceProperties [GT04] defines a mechanism by which Web Services can describe and publish *resource properties*, or sets of information about a resource. Resource property types are defined in the service's WSDL, and the resource properties themselves can be retrieved, in the form of XML documents, using WS-ResourceProperties query operations.
- WS-BaseNotification [GS04] defines a subscription/notification interface for accessing resource property information.
- WS-ServiceGroup [MS04] defines a mechanism for grouping related resources and/or services together as *service groups*.

The Index and Trigger services make extensive use of these standards and the mechanisms defined by them: both use service groups as part of their administrative interface, to keep track of what information they are to collect. The primary client interfaces to the Index Service are resource property queries and subscription/notification.

2.2 MDS4 Services

The central component in MDS is the Index Service, which collects information about Grid resources and makes this information available. It is similar to a UDDI registry [UDDI], however it does not have the static limitations of that approach, and allows the last value for every data element to be cached in order to improve query performance. The Index Service interacts with data sources via standard WSRF resource property and subscription/notification interfaces (WS-ResourceProperties and WS-BaseNotification). Any WSRF-based service can make information available as resource properties, however the Index Service collects information from (potentially) many sources and publishes it in one place. Resource properties may be queried by name or via XPath [XPATH] queries.

Administration of the Index Service is done via service groups (WS-ServiceGroup); service group entries describe the mechanisms and associated parameters used to collect data and to hold the collected data itself. This interface and the available data collection mechanisms are described in Section 2.3.

There may be many Indexes available to a Grid user. Each GT4 container has a default Index Service that keeps track of resources that have been created within the container. In addition, sites and virtual organizations often keep track of all the containers, resources, and services that are available to the site or VO in an Index Service. Users need only know the location of a single suitable Index Service in order to discover and monitor all of the resources and services that it indexes.

MDS4 Index Services have a number of features that are sometimes surprising to new users, but are necessary due to Grid scalability and policy issues:

- **Index Services can be arranged in hierarchies, but there is no single global Index that provides information about every resource on the Grid.** This is deliberate, as each virtual organization will have different policies on who can access its resources. No person in the world is part of every virtual organization!
- **The presence of a resource in an Index makes no guarantee about the availability of the resource for users of that Index.** An index provides an indication that certain resources are likely to be useful, but the ultimate decision about whether the resources can be used is left to direct negotiation between user and resource. A user who has decided on a particular service to access based on MDS information might still find they

are not authorized when they submit a job. This means that MDS does not need to keep track of policy information (something that is hard to do concisely) and that resources do not need to reveal their policies publicly.

- **MDS has a soft consistency model.** Published information is recent, but not guaranteed to be the absolute latest. This allows load caused by information updates to be reduced at the expense of having slightly older information. This delay is not a problem in practice – for example, it is generally acceptable to know the amount of free disk space on a system 5 minutes ago rather than 2 seconds ago.
- **Each registration into an Index Service is subject to soft-state lifetime management.** Registrations have expiry times and must be periodically renewed to indicate the continued existence of the resource. This allows each Index to be self-cleaning, with outdated entries disappearing automatically when they cease to renew their registrations.

In the most common use case, the Index Server essentially republishes data that was originally made available by some other service. Currently, however, the Index Server does not collect and enforce these remote servers' access control policies. To guard against the risk that an Index Server will allow broader access than the original publisher of the data intended, we recommend that the Index Servers be run in one of two modes: a *public index*, in which all Index data is collected through anonymous queries and access is granted to everyone, and a *personal index*, in which all index data is collected using credentials delegated by an individual and access is restricted to that same individual.

The *MDS-Trigger* service, the other higher-level service distributed as part of MDS4, collects information and compares that data against a set of conditions defined in a configuration file. When a condition is met, an action takes place, such as emailing a system administrator when the disk space on a server reaches a threshold. This functionality, inspired by a similar capacity in Hawkeye [Haw], has proven useful in trouble shooting for projects such as the Earth Science Grid (ESG) [BBB+05], who used the GT3 version of this service and are in the process of transitioning to the new software

2.3 Aggregator Framework Implementation

The MDS-Index and MDS-Trigger service implementations are both specializations of a more general *aggregator framework*, a software framework for building services that collect and aggregate data. Services built on this framework are sometimes called *aggregator services*. Such services have three properties in common.

First, they collect information via *aggregator sources*. An aggregator source is a Java class that implements an interface (defined as part of the aggregator framework) to collect XML-formatted data. MDS4 supports three types of aggregator source. A *Query* source uses WS-ResourceProperty mechanisms to poll a WSRF service for resource property information. A *Subscription* source collects data from a service via WS-Notification subscription/notification. Finally, an *Execution* source executes an administrator-supplied program to collect information. Figure 1 summarizes how information flows through the aggregator framework.

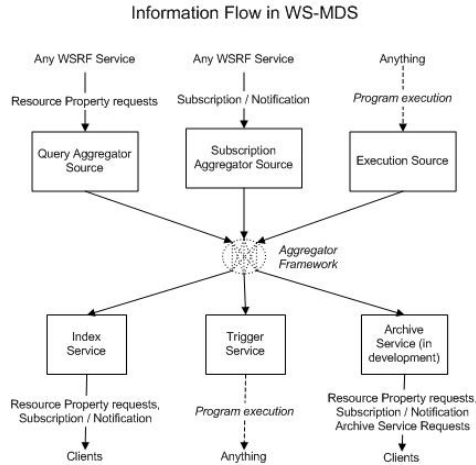


Figure 1: Information flow through the MDS4 aggregator framework.

Second, aggregator services use a common configuration mechanism to maintain information about which aggregator sources to use and their associated parameters, which generally specify what data to get, and from where. The aggregator framework WSDL defines an aggregating WS-ServiceGroup entry type that holds both configuration information and data. Administrative client programs use standard WS-ServiceGroup registration mechanisms to register these service group entries to the Aggregator Service.

Third, aggregator services are self-cleaning – each registration has a lifetime; if a registration expires without being refreshed, it and its associated data are removed from the server.

2.4 Information Providers

The data that an MDS4 aggregator source publishes into its aggregator service is always obtained from an external component called an *information provider*. In the case of a Query or Subscription source, the information provider is a WSRF-compliant service from which data is obtained via WS-ResourceProperty or WS-Notification mechanisms, respectively. In the case of an Execution source, the information provider is an executable program that obtains data via some domain-specific mechanism.

We have implemented seven such information providers in GT4, as summarized in Table 1. For each provider we give a name, the resource or service for which data is provided, the type of aggregator source, and the information made available. In all cases the schema is a standard XML document. For Hawkeye, Ganglia and GRAM, we publish information using the XML mapping of the GLUE schema [GLUE].

2.5 WebMDS User Interface

WebMDS is a web-based interface to WSRF resource property information that can be used as a user-friendly front-end to the Index Service. WebMDS uses standard resource property requests to query resource property data and XSLT transforms [XSLT] to format and display them. Web site administrators can customize their own WebMDS deployments by using HTML form options and creating their own XSLT transforms.

Table 2: Information providers included in GT4.0

| Name | Info source | Source Type | Information Provided |
|-------------|---|---------------------|---|
| Hawkeye | Condor pool | Execution | Basic host data (name, ID), processor information, memory size, OS name and version, file system data, processor load data, and other basic Condor host data. |
| Ganglia | Cluster | Execution | Basic host data (name, ID), memory size, OS name and version, file system data, processor load data, and other basic cluster data. |
| GRAM | GT4 grid resource allocation and management service | Query, Subscription | Processor information, memory size, queue information, number of CPUs available and free, job count information, and some memory statistics |
| RFT | GT4 reliable file transfer service | Query, Subscription | RFT service status data, number of active transfers, transfer status, information about the resource running the service |
| CAS | GT4 community authorization service | Query, Subscription | Identifies the VO served by the service instance |
| RLS | GT4 replica location service | Execution | Location of replicas on physical storage systems (based on user registrations) for later queries. |
| Basic | Every GT4 Web service | Query, Subscription | ServiceMetaDataInfo element includes start time, version, and service type name |

2.6 Putting it all together

To describe a typical MDS4 deployment we envision a multi-project VO that consists of 30 sites (3 representative sites are shown in Figure 2), and a wide set of collaborating applications. These systems are heterogeneous in nature, and deploy a varied set of software and services.

Working from the local level up, each clustered resource has deployed Ganglia (for common queued clusters) or Hawkeye (for condor pools) for host-level monitoring and to allow MDS access to scheduler and cluster information. Specifically, in our summary picture, Site 1 has two clusters, each with a Ganglia deployment and Site 2 is running Condor and the Hawkeye monitoring tool. Note also that the two clusters at Site 1 are running different queuing systems (one has PBS; the other has LSF) this doesn't make a difference in our MDS deployment. More information about deploying the Ganglia or Hawkeye information provider to view cluster information in the Index Service is also available [MDSa].

Each site is also running additional services. Shown in the figure is Site 1 is running an RFT server and Site 3 is running an RLS server.

Each site has also deployed a site-wide Index service (the one for Site 1 is labeled A in Figure 2). This has all the services and resources at the site registered to it, and will allow each site to view its local resources, including those provided by Ganglia or Hawkeye [MDSb].

Application B also has an application-specific Index set up (labeled B in Figure 2) which has registrations for the application specific services, in this case the RFT server at Site 1 and the RLS server at Site 3. This allows those application users to easily see just those resources and services specific to that application.

This project has decided on a 3-level tiering for the VO-wide indexes. The first tier is at the site level, as described. The second tier is an East Coast-West Coast division, where Sites 2 and 3 share a combined West Coast Index running at Site 2 (labeled C in Figure 2). Site 2 also maintains the VO-wide server running on a resource at Site 2 (labeled D in Figure 2) to which the Index servers from the other sites have registered as well. This allows anyone from the VO to view all of the resources available to the full VO [MDSc]. In general this hierarchy can be arbitrarily deep.

The VO has deployed WebMDS as well (labeled D in Figure 2) so all VO users can view the current state of the resources and services across the VO [MDSd]. They have also deployed a Trigger Service (not shown) to alert interested parties about changes in the status of the VO [MDSe]. The VO operations center uses this to be automatically advised of failures in services.

With this deployment, members of the project can discover needed data from services in order to make job submission or replica selection decisions by querying the VO-wide Index; evaluate the status of Grid services by looking at the VO-wide WebMDS setup; be notified when disks are full or other error conditions happen by being on the list of administrators, part of the Trigger Service set up. Individual projects can examine just the state of the resources and services of interest to them, as Application B is doing.

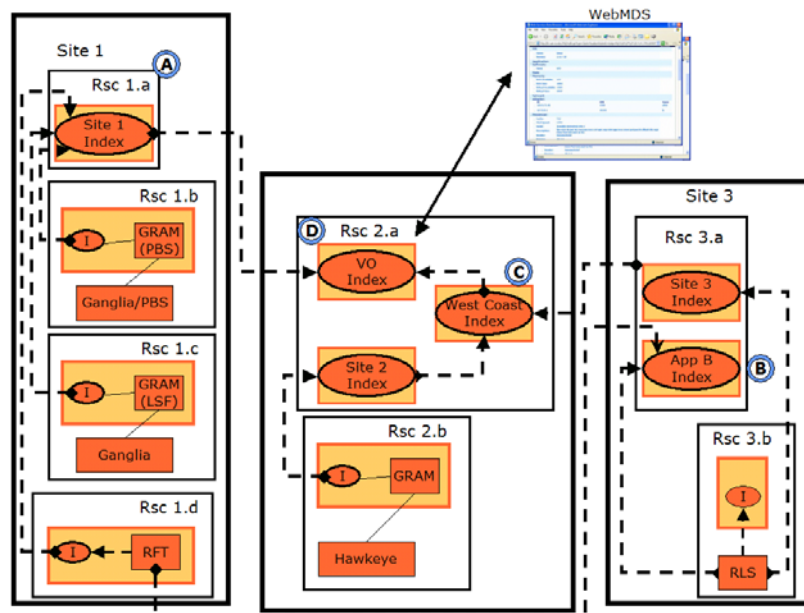


Figure 2. Sample MDS4 deployment. Orange (grey) box is a container, white boxes with a small outline are resources, white boxes with a thick line stand for sites. The dashed lines are registrations, and ovals are Indexes.

3 MDS4 Performance Results

For any new software, some basic performance information is needed to determine its feasibility for use. In its simplest form, we believe this includes:

- 1) How long does one response take?
- 2) How many responses per minute are possible?
- 3) How long does the service stay up while being used before failing?

For MDS4 the most important service to investigate in terms of performance is the Index Service, so we began our performance analysis there. These experiments show results for the 3.9.5 version (beta to the 4.0 release) of MDS4. *For the final draft of the paper these experiments will be re-run using the 4.0 final code.*

For the following experiments, the testbed consisted of a set of 5 client nodes (ned0–ned4) and one server node (dc-user2) all located at ISI/USC. Each client node was a dual CPU 1133MHz Pentium III machine with 1.5GB of RAM. The server node was a dual Intel (hyperthreaded) Xeon running at 2.20GHz with 1GB of RAM. All of these machines are interconnected by Gigabit Ethernet and are located on the same physical switch. Dedicated access to the client nodes was obtained for the duration of these tests, however the server was shared during the experiments, although the monitored load was not substantially high during the testing.

When we ran experiments using 25 clients, we ran 5 processes on each of 5 client machines. For testing the performance of 100 clients, we ran 20 processes on each of 5 client nodes.

The 'entries' described in these tests are pieces of data that are registered with the Index Service using the standard mds-servicegroup-add tool included in the GT4 release. Each entry consisted of all standard pieces of information required for a ServiceGroupEntry (i.e. ServiceGroupEntryEPR, MemberServiceEPR, Aggregator configuration information), as well as a small amount of filler data that simulated useful information. A simple script that published formatted data of a constant size generated the filler data, which was then registered to the Index Service as a simple Execution Aggregator Source. As the number of entries increased, the size of the queried data increased. Each entry returned by a single query was approximately 1.9KB (1850 bytes) in size. Thus, a single query for 100 entries returned approximately 190KB of data across the network.

All registered entries had an identical configuration in that the scripts were executed by the Index Service every 10 minutes, thereby regenerating the cached data. Due to the soft state nature of the registration, the registration itself was also configured to be renewed with the Index Service every 10 minutes. These values are reasonable defaults for actual data published in a Web service-based system, although current Grid-level service monitoring seen in other projects are more on the order of 1 hour, 6 hour, or 24 hour updates for service uptime checks [GITS] [SOE+04]. The duration of each test lasted well over 10 minutes to ensure that the updates would occur several times.

For the test results presented, it is important to recognize that we have averaged aggregate data across all of the involved clients. Not all clients performed equally on any given machine. In the tests with larger numbers of client processes, it should be noted that the results between clients varied quite a bit. For example, in one run with 100 client processes for 100 entries, a client on one machine was obtaining an average query response time of ~1000 milliseconds, while another querying the same index server had an average query response time of ~7000 milliseconds. Thus, the average across all clients is used to indicate an average expected performance.

3.1 Index Service Performance

Our first set of experiments study the impact of index size on query performance. We ran repeated sequential queries from 1, 2, 25, or 100 clients on one machine against one service on the server machine, as detailed above. Table 3 summarizes our results for different index sizes.

Table 3: Index service per client performance.

| Index Entries | 1 client | | 2 Clients | | 25 Clients | | 100 Clients | |
|---------------|----------------------------|----------------------|----------------------------|----------------------|----------------------------|----------------------|----------------------------|----------------------|
| | Single client queries /sec | Response Time (msec) | Single client queries /sec | Response Time (msec) | Single client queries /sec | Response Time (msec) | Single client queries /sec | Response Time (msec) |
| 10 | 24 | 40 | 22 | 44 | 4.5 | 245 | 0.85 | 1243 |
| 30 | 15 | 64 | 10 | 93 | n/a | n/a | n/a | n/a |
| 100 | 5 | 190 | 4 | 265 | 0.78 | 1334 | 0.19 | 5824 |

As expected, as the MDS4 Index grows, query rate and response time both slow, although sublinearly. We believe the response time slows due to increasing data transfer rate, since the full Index is being returned. The response is re-built for every query and re-sent each time, and as the number of entries increase the packaging and message size grows. This can also be seen when we compare these results to the stability experiment, which has a 0-entry Index and even better performance. *In the final draft of the paper we will be able to show results for 25 and 100 clients for our 30-entry Index test. We will also use NetLogger to further investigate which phase of the query is the performance bottleneck.*

3.2 MDS4 Index Compared to non-Web Service Approaches

Table 4 shows a rough comparison of MDS4 index queries as described versus 4 other Grid monitoring systems: MDS2 (v. 2.4.3) with caching, MDS2 (v.2.4.3) without caching, R-GMA [CGM+03] version 3.4.6, and Hawkeye (version 1.0). This data came from a related paper [ZFS05] and the comparison is not quite equal. In the other experiments, the size of the index was 10 entries, and the query was for the full set of values, but that varied from 10KB (for MDS2 and Hawkeye) to 2KB (for R-GMA), compared with approximately 19KB for MDS4. The testbed setup was also slightly different, although both client and server machines were similar, the network was between LANs, with a bandwidth of approximately 55 Mbits per sec on average and a latency (Round-Trip Time) of 2.3 msec on average. Since a different number of clients were run in those experiments and ours, we have approximated the missing values for MDS4 using simple curve fitting, these values are in italics.

Table 4: comparison of Index service performance for 5 monitoring systems. Note values in italics are curve-fitting approximations.

| Monitoring System | 1 client | | 10 Clients | | 50 Clients | | 100 Clients | |
|-------------------|----------------------------|----------------------|----------------------------|----------------------|----------------------------|----------------------|----------------------------|----------------------|
| | Single client queries /sec | Response Time (msec) | Single client queries /sec | Response Time (msec) | Single client queries /sec | Response Time (msec) | Single client queries /sec | Response Time (msec) |
| MDS2 w/cache | 0.88 | 129 | 0.45 | 147 | 0.92 | 153 | 0.93 | 182 |
| MDS2 w/o cache | 0.45 | 1219 | 0.15 | 5534 | 0.77 | 29,175 | 0.91 | 40,410 |
| R-GMA | 0.92 | 61 | 0.03 | 277 | 0.24 | 3230 | 0.89 | 9734 |
| Hawkeye | 0.93 | 79 | 0.02 | 106 | 0.12 | 113 | 0.68 | 463 |
| MDS-4 | 24 | 40 | <i>16.8</i> | n/a | <i>3.29</i> | n/a | 0.85 | 1243 |

We plan to have an apples-apples comparison for the other monitoring system data for the final draft of the paper. We would also like to explore how our performance compares to that of basic WSRF/WSN.

3.3 Index Service Stability

We set up an Index Service with only one entry and ran queries against it over a longer period of time to help judge the stability of the service. After running for 1,225,221 seconds (just over 2 weeks), the server machine was accidentally rebooted. In that time, the Index service processed 93,890,248 requests, averaging 76 per second, with an average query round-trip time of 13 milliseconds.

We feel this indicates that the Index Service is a stable service, as there was no noticeable performance or usability degradation over the entire duration of the test. There was also no indication that the test would not continue indefinitely had it been run for a longer period of time.

4 Related Work

Performance studies of Grid monitoring systems include work on previous MDS versions. Smith et al. [SWM+00] investigated MDS2 performance by focusing on the effect of different versions of backend LDAP and data distribution strategies. Aloisio et al. [ACE+01] studied the capabilities and limitations of MDS2 as well as the security effect on the performance although their experiments were limited to simple tests on the MDS2 Index (GIIS) only.

Keung et al. [KDJ+03] analyzed the MDS2 GRIS performance with different back-end implementations by varying information-gathering methods. More recently, Keung et al. [KDJ+03b] evaluated performance differences when querying the MDS2 GIIS using different evaluation methods. This work compliments our studies [ZS04] in which we examined MDS2 behavior at a fine granularity by using NetLogger technologies to instrument the server and client codes, but did not compare this behavior to any other system.

In more general studies, Plale et al. [PJJ+04, PJJ+03] benchmarked a synthetic workload (queries and updates) against a non realistic Grid information service implemented with three different database platforms: relational (MySQL), native XML (Xindice), and LDAP. In other work, Plale et al. [PDvL02] discussed the pros and cons of building a Grid Information Service on a hierarchical representation and a relational representation; however, their approach was theoretical not experimental. We have also examined the scalability performance of MDS2, R-GMA, and Hawkeye in a coarse-grain manner [ZFS03], and then in more detail using NetLogger to better understand the performance benchmarks [ZFS05].

5 Conclusion

In this paper we have shown by example how monitoring and discovery capabilities can be integrated into the design of a distributed computing infrastructure so that any and every resource and service can be monitored and discovered in a uniform manner. Using Web Service standards that define the primitive interfaces and behaviors, we have built the basis of a monitoring system for Grid use. Our initial performance results indicate that the basic performance is acceptable, although further work is needed to understand performance bottlenecks of the system.

Acknowledgments

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract W-31-109-Eng-38. Additional support was provided by NSF NMI Award SCI-0438372.

References

[ACE+01] G. Aloisio, M. Cafaro, I. Epicoco, and S. Fiore, "Analysis of the globus toolkit grid information service," GridLab, technical report GridLab-10-D.1-0001-GIS_Analysis, 2001

[BBB+05] Bernholdt, D., Bharathi, S., Brown, D., Chanchio, K., Chen, M., Chervenak, A., Cinquini, L., Drach, B., Foster, I., Fox, P., Garcia, J., Kesselman, C., Markel, R., Middleton, D., Nefedova, V., Pouchard, L., Shoshani, A., Sim, A., Strand, G. and Williams, D. The Earth System Grid: Supporting the Next Generation of Climate Modeling Research. Proceedings of the IEEE, 93 (3). 485-495. 2005.

[CFF01] Czajkowski, K., Fitzgerald, S., Foster, I. and Kesselman, C., Grid Information Services for Distributed Resource Sharing. 10th IEEE International Symposium on High Performance Distributed Computing, 2001, IEEE Computer Society Press, 181-184.

[CGM+03] A. Cooke, A.Gray, L. Ma, W. Nutt, J. Magowan, P. Taylor, R. Byrom, L. Field, S. Hicks, and J. Leake, "R-GMA: An Information Integration System for Grid Monitoring," Proceedings of the 11th International Conference on Cooperative Information Systems, 2003.

[Foster05] I. Foster, A Globus Toolkit Primer, www.globus.org/primer, 2005.

[FCF+05] Foster, I., Czajkowski, K., Ferguson, D., Frey, J., Graham, S., Maguire, T., Snelling, D. and Tuecke, S. Modeling and Managing State in Distributed Systems: The Role of OGSI and WSRF. Proceedings of the IEEE, 93 (3). 604-612. 2005.

[GITS] Grid Integration test Service (GITS), <http://www.ngs.rl.ac.uk/sites/common/docs/gits14.html>

[GLUE] Glue Schema Specification, www.hicb.org/glue/glue-schema/schema.html.

[GM04] Steve Graham, Bryan Murray, eds., "Web Services Base Notification 1.2 (WS-BaseNotification)", OASIS Working Draft #wsn-WS-BaseNotification-1.2-draft-03, June 2004, <http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf>

[GT03] D. Gunter and B. Tierney, "NetLogger: A Toolkit for Distributed System Performance Tuning and Debugging," Proceedings of Integrated Network Management 2003, 2003.

[GT04] Steve Graham, Jem Treadwell, eds., "Web Services Resource Properties (WS-Resource Properties)", OASIS Working Draft #wsrf-WS-ResourceProperties-1.2-draft-04, June 2004, <http://docs.oasis-open.org/wsrp/2004/06/wsrp-WS-ResourceProperties-1.2-draft-04.pdf>

[Haw] Hawkeye, <http://www.cs.wisc.edu/condor/hawkeye>.

[KDJ+03] H. N. Lim Choi Keung, J. R. D. Dyson, S. A. Jarvis, and G. R. Nudd, "Performance evaluation of a grid resource monitoring and discovery service," *IEEE Proceedings: Software*, vol. 150, pp. 243-251, 2003.

[KDJ+03b] H. N. Lim Choi Keung, J. R. D. Dyson, S. A. Jarvis, and G. R. Nudd, "Predicting the Performance of Globus Monitoring and Discovery Service (MDS-2) Queries," Proceedings of the 4th International Workshop on Grid Computing, 2003.

[MCC04] Massie, M.L., Chun, B.N. and Culler, D.E. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30 (7). 2004.

[MDSa] "GT 4.0 WS MDS: Cluster Monitoring Information and the GLUE Resource

Property", <http://www-unix.globus.org/toolkit/docs/development/4.0-drafts/info/key/gluerp.html>

[MDSb] "GT 4.0 WS MDS Index Service: System Administrator's Guide",
<http://www-unix.globus.org/toolkit/docs/development/4.0-drafts/info/index/admin/>

[MDSc] "Deploying WS MDS in a Virtual Organization",
http://www-unix.globus.org/toolkit/docs/development/4.0-drafts/info/key/deployment_overview.html

[MDSd] "GT 4.0 WS MDS WebMDS",
<http://www-unix.globus.org/toolkit/docs/development/4.0-drafts/info/webmds/>

[MDS e] "GT 4.0 WS MDS Trigger Service: System Administrator's Guide",
<http://www-unix.globus.org/toolkit/docs/development/4.0-drafts/info/trigger/admin/>

[MS04] Tom Maguire, David Snelling, eds., "Web Services Service Group 1.2 (WS-ServiceGroup)", OASIS Working Draft #wsrf-WS-ServiceGroup-1.2-draft-02, June 2004, <http://docs.oasis-open.org/wsr/2004/06/wsr-WS-ServiceGroup-1.2-draft-02.pdf>

[PDvL02] B. Plale, P. Dinda, and G. v. Laszewski, "Key concepts and services of a grid information service," Proceedings of the 15th International Conference on Parallel and Distributed Computing Systmes (PDCS), 2002.

[PJJ+04] B. Plale, C. Jacobs, S. Jensen, Y. Liu, C. Moad, R. Parab, and P. Vaidya, "Understanding Grid Resource Information Management through a Synthetic Database Benchmark/Workload," Proceedings of the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004) (to appear), 2004.

[PJL+03] B. Plale, C. Jacobs, Y. Liu, C. Moad, R. Parab, and P. Vaidya, "Benchmark Details of Synthetic Database Benchmark/Workload for Grid Resource Information," Indiana University Computer Science Technical Report TR-583 Technical Report TR-583, August 2003 2003.

[SOE+04] Shava Smallen, Catherine Olschanowsky, Kate Ericson, Pete Beckman, and Jennifer Schopf, "The Inca Test Harness and Reporting Framework", Proceedings of SuperComputing '04, November 2004. Also available as SDSC Technical Report #SDSC-TR-2004-3, <http://www.sdsc.edu/TR/SDSC-TR-2004-3-IncaTest.pdf>.

[SWM+00] W. Smith, A. Waheed, D. Meyers, and J. Yan, "An Evaluation of alternative designs for a grid information service," Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC-9), 2000.

[UDDI] UDDI Standard, <http://www.uddi.org>

[XPATH] XML Path Language (XPath) Version 1.0, Nov. 1999, <http://www.w3.org/TR/xpath>

[XSLT] XSL Transformations (XSLT) Version 1.0, Nov. 1999, <http://www.w3.org/TR/xslt>

[ZFS03] X. Zhang, J. Freschl, and J. M. Schopf, "A performance study of monitoring and information services for distributed systems," Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12), 2003.

[ZS04] X. Zhang and J. M. Schopf, "Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2," Proceedings of IEEE IPCCC International Workshop on Middleware Performance (IWMP 2004), 2004.

[ZFS05] Xuehai Zhang, Jeffrey L. Freschl, and Jennifer M. Schopf, "A Scalability Analysis of Three Monitoring and Information Systems: MDS2, R-GMA, and Hawkeye", ANL Tech Report, available from www.mcs.anl.gov/~jms/jmspubs.html, 2005.